# Design of a Processor to Support the Teaching of Computer Systems

Murray Pearson, Dean Armstrong and Tony McGregor
Department of Computer Science
University of Waikato
Hamilton
New Zealand
{mpearson,daa1,tonym}@cs.waikato.nz

## Abstract

*Teaching computer systems, including computer architecture, assembly language programming and operating system implementation, is a challenging occupation. At the University of Waikato this is made doubly true because we require all computer science and information systems students study this material at second year. The challenges of teaching difficult material to a wide range of students have driven us to find ways of making the material more accessible. The corner stone of our strategy for delivering this material is the design and implementation of a custom CPU that meets the needs of teaching. This paper describes our motivation and these needs. We present the CPU and board design and describe the implementation of the CPU in an FPGA. The paper also includes some reflections on the use of a real CPU rather than a simulation environment. We conclude with a discussion of how the CPU can be used for advanced classes in computer architecture and a description of the current status of the project.*

## 1 Introduction

Teaching computer systems is a challenging but vital part of the computer science curriculum. In 1997 the Department of Computer Science at the University of Waikato decided that computer systems was important to all computer science and information science students and made its computer systems course compulsory for all second year students. Like most computer systems courses Waikato's uses assembly language programming as a vehicle to understanding the interrelationships and interactions between the different components of a computer system. While very few of the students will continue to program in assembly language after the course we believe that it is important that they have an understanding of computer operation at this level of abstraction. Unfortunately many students find assembly language programming difficult and this detracts from the main thrust of the course, which is not to teach assembly language per say.

Advances in reconfigurable logic have opened new possibilities for teaching courses in the computer systems area. We have used this technology to develop a single board computer with with our own custom designed CPU and IO devices to support teaching of computer systems. By designing our own CPU, using FPGA technology, we are able to use an instruction set that is optimised for simplicity rather than performance. By not having to spend a great deal of time talking about performance optimising features we are able to spent this time giving a broader coverage of the subject area.

In the next section a brief course outline for our computer systems course is given. Section 3 then describes in more detail the motivation for developing a processor and board to support the teaching this course. Sections 4 and 5 describe the design of the CPU and board. A brief description is then given of how the we intend to use the board in the third and fourth year computer architecture courses.

## 2 Course Outline

When the Department decided to make the second year computer systems course compulsory its curriculum committee established a set of key topics that should be covered by the course. These included data representation, machine architecture (including assembly language programming), memory and IO, operating systems and data communications.

Figure 1 shows the order of the topics that make up the course and the relative levels of abstraction used to describe them. The main content of the course can be broken into two parts. The first part illustrates what happens to a high level program when it is compiled and executed on a computer system. This serves two purposes. First, it demon-

Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications (DELTA'02)
0-7695-1453-7/02 $17.00 © 2002 IEEE

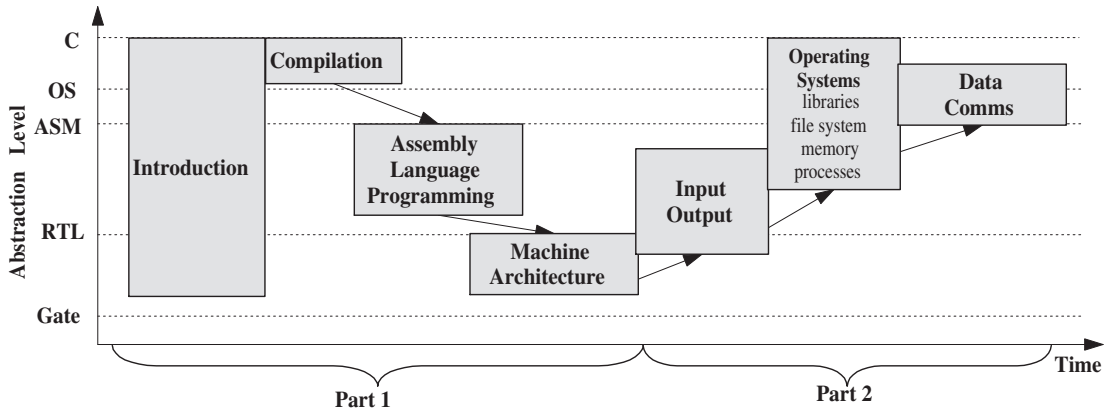



**Figure 1. Topics Covered in the Course**

strates some of the major issues which determine the performance of a computer system. Second, it shows the likely consequences of writing a particular construct in a high level programming language in terms of speed and size of the code generated.

The aim of the second part of the course is to produce an understanding of operating system principles and components, their role in supporting the user, and in the execution of programs written in high level languages such as C (the starting point of the course). The focus is on achieving an empathy with the operating system rather than an ability to write a new one.

We believe that the best way to meet the aims set for the course is to base the whole course around a single processor architecture so that the students could more easily see the way the individual components of the system contribute to the complete computer system.

## 3 Background

Because the goal of the course is to explain the role and interaction of the components of a computer system, not to teach assembly language programming for its own sake, there are two main requirements for a model architecture:

1. a simple, easy to learn instruction set
2. an architecture that can easily demonstrate the relationship between high and low level languages.

These goals are at odds with most modern CPU architectures which have been optimised to maximise performance and not simplicity. To help achieve these very high levels of performance these modern CPUs contain many performance oriented techniques. These include the use of reorder buffers, register renaming and reservation stations [6]. Because of the complexity of these architectures it would not be possible to fully describe the structure and functionality of one of them in an introductory course.

While most architectures are optimised for performance some (such as the 8 bit processors like the Motorola HC11) are designed to be very cheap and simple. However, in some cases this very simplicity raises the complexity required to program the CPU. For example, performing 16-bit indexed address access on an 8-bit processor that only has an 8-bit ALU requires a series of instructions to support the 16 bit addition rather than the single instruction available on larger word sized machine. Because of the way CPUs developed through the late 80's and early 90's CPUs with a large enough word size to make those aspects of programming easy, have other complexities, such as many addressing modes that are not available across all instructions or complex interrupt processing. Although many modern CPUs are simpler, because if the influence of the RISC philosophy, they have other disadvantages, including branch and load delays as described below.

In the past, we have used the MIPS R3000 family as a compromise between the needs of our course and available CPU designs [4]. The MIPS CPUs have a relatively simple programmer's abstraction. The teaching process is also supported by a number of very popular text books including those written by Henessey and Patterson [3] [2] and Goodman and Millar [1]. For this reason our computer systems course has been based around this processor for the last six years. While we have found this processor reasonably well suited to our needs, we have identified a number of issues with the architecture that a large body of the students find difficult to understand and which are not central to our teaching goals. These include:

- the presence of *load delay slots* which mean that the instruction directly after a load instruction cannot use the result of the load as it isn't available yet.

Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications (DELTA'02)
0-7695-1453-7/02 $17.00 © 2002 IEEE

- the presence of *branch delay slots* which mean that the instruction directly after a branch instruction is always executed regardless of whether the branch is taken or not.
- the use of an *intelligent assembler* which is capable of reordering instructions and breaking some assembler instructions in two so that they can all be encoded using a single 32-bit word.
- the requirement that all *memory accesses to word values are word aligned*.
- the *parameter passing conventions* that are designed to minimise the number of stack manipulations in a MIPS program.

While we do not believe that the complexities described above are insurmountable they do detract from the goals of the course, that is to give a complete coverage of the computer systems area at an introductory level without being distracted by the complexities associated with describing a particular manufacturers quirks. This is in keeping with the introductory level and broad audience that this course is intended for. Other courses at the University are intended for students who will specialise in computer architecture and do cover commercial architectures, including exposure to many of these issues.

We have been unable to find a suitable commercial CPU architecture to support the teaching of our computer systems course so we developed our own.

Before discussing the architecture of the CPU we have designed we consider the question of whether to use a real CPU or a simulator. Most courses that teach computer architecture or assembly language teaching make use of CPU simulators. Using a simulated system offers two main advantages. Firstly, it is possible to develop a simulator for any CPU. This allows a CPU that is tailored to the goals of the course to be used rather than being limited to those that are available commercially. The second advantage of using a simulator is that simulators normally offer better debugging and visualisations of a program. These can be used to help reinforce important concepts.

Although using a simulator offers advantages, a simulator is itself a program running on a computer. This makes if difficult for students to readily identify the target system and may confuse the role of components of the system. When this happens there is a risk that students will focus on the most obvious difference between practical work in this area and others: the programming language. The use of real hardware makes the distinctions between the target system and the development tools clear and strengthens the model students have of the system they are working with. We believe that making the conceptual environment simpler outweighs the disadvantages of using a real CPU. The work presented in this paper largely removes these disadvantages and enables both the simpler working model and a CPU designed to meet the needs of teaching that has good debugging and the ability to 'see into' the system as it executes.

## 4 Processor Design

In designing a processor a great deal of care has been taken to keep the design as simple and regular as possible while still being able support the complete range of practical experiences we wish the students to be exposed to. These experiences start with the writing of simple assembly language programs and build up to the development of a very simple multi-tasking kernel.

The resulting CPU design uses a 32 bit word, and is based around a register-register load-store architecture, very similar to the MIPS and DLX [5] processors. Most computational instructions have a three operand format, where the target and first source are general purpose registers, and the second source is either a register or an immediate value. Regularity of the instruction set was a key factor in maintaining the simplicity. Immediate flavours of all computational instructions are provided, as well as unsigned versions of all arithmetic instructions.

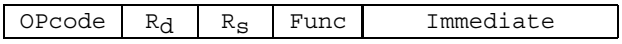
Care was taken to keep the correspondence between assembly language instructions and actual machine instructions as a one-to-one relationship. To this end a major feature of this CPU is the reduction of the address width to 20 bits, and the number of registers to 16. This allows an address, along with two register identifiers and an opcode to fit into a single instruction word, removing the need for assembler translation when a program label is referenced.

The other main differences from MIPS and DLX are the removal of the branch and load delay slots, and the fact that the CPU is 32 bit word addressable rather than byte addressable. Making the machine word addressable only, greatly simplifies the operation of the CPU, and allows us to present students with an easily understandable model of it. Another advantage of a word addressable machine is that it removes the possibility of word access alignment problems which new students frequently encounter on a byte addressable machine.

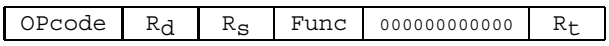
The CPU only supports three instruction formats as shown in Figure 2. It can also be seen from this figure that the instructions have been encoded to allow for easy manual disassembly from a hexadecimal number, with all fields aligned on 4 bit boundaries.

While the CPU has been made as simple as possible for the tasks we require it does include external and software interrupts and has supervisor and user modes with protection. These mechanisms are accessed through a special register file, similar to the MIPS' coprocessor 0. This means that

**I-Type instruction**




**R-Type instruction**




**J-Type instruction**


| | |
|---|---|
| OPCode | 4 bit operation code |
| $R_d$ | 4 bit destination register specifier |
| $R_s$ | 4 bit source register specifier |
| $R_t$ | 4 bit source register specifier |
| Func | 4 bit function specifier |
| Immediate | 16 bit immediate field |
| Address | 20 bit address field |

**Figure 2. Instruction encoding formats**

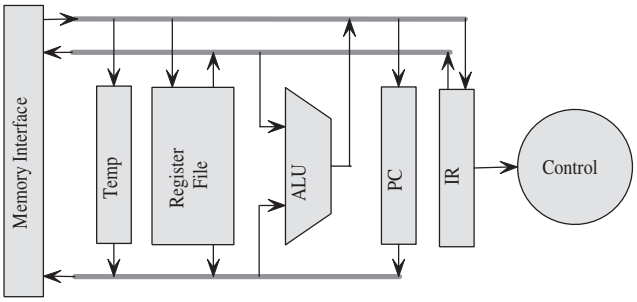



**Figure 3. Processor Block Diagram**

these concepts need not be discussed for students to begin programming in assembler, and when desired, they can be introduced by describing the special register file, and the two instructions needed to access its contents.

The data-path of the processor is based around a three-bus structure (as shown in Figure 3) and instructions take multiple clock cycles to execute. As can be seen from Figure 3 the CPU's data-path is very simple making it possible to completely explain the operation of the data-path. In particular it is possible to explain in detail how machine code instructions stored in memory can be fetched, decoded and executed on the data-path.

The CPU has been represented in VHDL so that it can be targeted to a reconfigurable logic device. The CPU design when synthesised consumes most of a 100 thousand gate Xilinx Spartan II FPGA device.

## 5 Board Design

Figure 4 shows a photograph of the printed circuit board designed to support the CPU described in the previous section. As can be seen from the picture we have been careful to layout the board so that the main components that make up a computer system can be clearly identified. The main data-paths that connect these components are also visible on the board.

Reconfigurable logic is used wherever possible on the board to allow it to be as flexible as possible. In addition to making the design of our own CPU and IO devices possible, this allows the architecture of these components that students are presented with to be fine tuned as the course develops. As explained later, it also allows the board to be used for multiple teaching functions, including FPGA and CPU design.

While it would have been possible to place most or all of the reconfigurable designs into a single chip the decision was made to use a separate chip for each IO device and the CPU, making it possible for the students to physically identify each of these devices on the board. The choice to use multiple RAM and ROM chips to provide the 32 bits of data rather than employing multiple accesses to a single chip was also made with the intention of clarifying the operation for the students. Effort was made, however to keep the number of non-essential support components to a minimum.

The boards are intended to be connected to a workstation where students can write and assemble programs, which can then be loaded and run on the board. Because we want to build a laboratory for a large class it was important to make reconfiguration easy. In particular we designed the board to support remote reconfiguration of all programmable devices and the stored bootstrap program code. This means that a program can be remotely run on each workstation connected to a board, which reconfigures that board. Cost has also been kept to a reasonable level.

Although there are a number of features that support teaching, one that had a large impact on both the board and CPU design is support for cycle-by-cycle stepping of the processor with an LCD display to indicate bus contents, and LEDs to show device selection and exceptions. We believe this feature will be a major asset for students struggling with the many new abstractions and concepts presented by the course.

## 6 Use of the Board by 3rd and 4th year Students

Next year, we plan to start teaching students in the third year computer architecture course about design using VHDL. By the end of the course it is hoped that the students

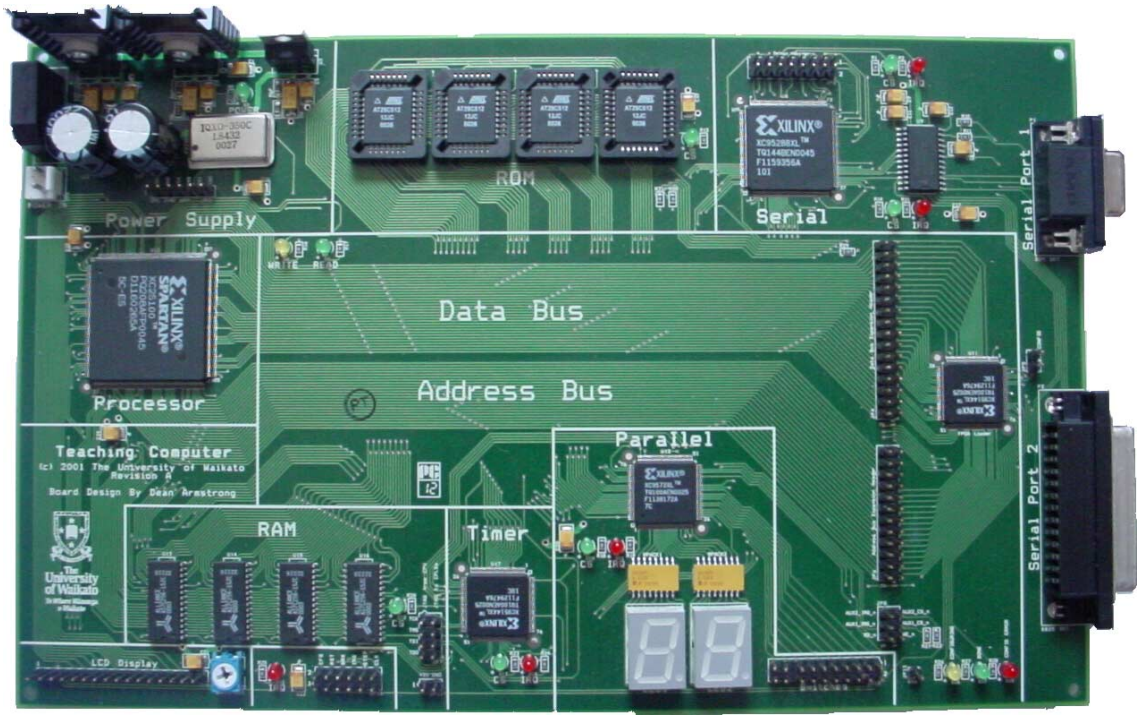



**Figure 4. Photo of the Board**

will be able to design the main components (ALU, registers, finite state machines, etc) that make up a CPU.

Currently, in our fourth year computer architecture course, students design and implement their own CPU on a board containing an FPGA that we developed several years ago. With the introduction of the new board and the experience gained using the board in the second and third year courses, we hope to be able to extend the complexity of the project undertaken in this course.

## 7 Conclusions

There is much merit in the design of custom CPU and IO devices for teaching purposes. Current reconfigurable hardware devices have made it possible to build a single board computer, with a custom CPU and IO devices, to support the teaching of computer systems courses at the University of Waikato. Using this approach we have removed some of the 'sharp edges' of assembly language programming, like branch delay slots and complex CPU status control, that add complexity to introductory teaching but do not add significant value. An additional advantage is that the board will provide a consistent teaching platform across a range of courses. We expect that this will considerably enhance the students learning experience.

At present a prototype version of the board and CPU design has been completed. We are now starting to turn our attention to the development of supporting tools such as a compiler and monitor for the board. We expect to start teaching using the board in the 2002 academic year.

## References

[1] J. Goodman and K. Millar. *A Programmer's View of Computer Architecture with Assembly Language examples from the MIPS RISC Architecture*. Oxford Press, 1992.

[2] J. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, Second Edition*. Morgan-Kaufman, 1995.

[3] D. A. Patterson and J. Hennessy. *Computer Organisation and Design: The Hardware/Software interface*. Morgan-Kaufman, 1994.

[4] M. Pearson, A. McGregor, and G. Holmes. Teaching computer systems to majors: A MIPS based approach. *IEEE Computer Society Computer Architecture Technical Committee News Letter*, pages 22–24, Feb. 1999.

[5] P. M. Sailer, P. M. Sailer, and D. R. Kaeli. *The DLX Instruction Set Architecture Handbook*. Morgan-Kaufmann, 1996.

[6] R. M. Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. In *IBM Journal of Research and Development*, volume 11, pages 25–33. 1967.